In *Proceedings of the Fifteenth Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

Managing decision resources in plan execution

Michael Freed and Roger Remington
NASA Ames Research Center, MS 262-4
Moffett Field, CA 94035 USA

Abstract

We describe an approach to the problem of managing resources in routine decision-making tasks. The central feature of this approach is the use of reusable RAP-like plans to generate decisions. This allows our system, APEX, to take advantage of the flexibility in scheduling and method selection provided by execution mechanisms and thereby minimize or circumvent resource conflicts. We then discuss an application of APEX for simulating a human air traffic controller in order to aid in the evaluation of radar display designs.

1 Introduction

In this paper, we describe an approach to managing resources in *routine decision tasks* and apply this approach to a practical problem. Routine decisions are choices that occur regularly in an agent's everyday tasks. For example, drivers are often faced with decisions such as whether to slow down for a yellow light and whether to turn at an often-encountered intersection. Making such decisions involves several resource-demanding activities including acquiring decision-relevant information (internally or from the task environment) and making inferences. In time-pressured conditions, or when multiple tasks compete for the same computational and perceptual resources, the ability to manage scarce resources becomes an important determiner of agent performance.

Researchers have taken a variety of approaches to managing resources, especially computational resources, when deciding action in realistically complex, dynamic task environments. One approach is to eliminate certain expensive computations. Reactive planners, for example, use only current perceptions to conditionalize action choice, thus avoiding the expensive computations required to construct plans and retrieve items from memory. Some systems avoid specific classes of inference such as the prediction of future states and deductive retrieval [Firby89]. Others allow expensive operations when they are most likely to prove essential but otherwise avoid them [Chien91;Hayes-Roth95].

A second approach is to delay decisions until relevant information can be acquired cheaply (or at all), thus avoiding the computationally expensive process of conditionalizing decisions on a large number of possible future states. Systems that employ this approach [Firby89; Gat96; Simmons94; Pell97] are referred to as execution systems since they interleave planning (deciding action) with plan execution.

Our system's plan execution component, like the RAP system [Firby89] on which it is based, combines these approaches but differs from previous systems in how it handles routine decisions. In particular, routine decision tasks are treated in a uniform manner with tasks of other kinds — i.e. they are carried out by general-purpose task-execution mechanisms in accordance with task-specific, variablized plan structures called *methods*. Methods consist of steps, each corresponding either to a primitive action or to some non-primitive that must be decomposed into substeps by selecting a more specific method.

2 Decision-methods

The use of specialized decision-plans has a number of advantages. In particular, plan execution mechanisms can begin, abort, retry, interrupt, resume, specify and terminate decision tasks as needed to handle situational constraints and coordinate resource use with other tasks. Consider, for example, the method below for deciding between alternative routes home from work.

```
(method-25 (decide-route-from-work-to-home)
  (step1 (acquire-info:time-of-day => ?time)
       (priority +1))
  (step2 (acquire-info:day-of-week => ?day))
  (step3 (infer rule-57 ?time ?day => ?best-route)
       (wait-for step1 step2)))
```

In this case, the decision is based solely on whether or not the agent is likely to face rush hour traffic, a function of the time of day and day of week. The two information acquisition steps are carried out in parallel, each by one of several methods. For instance, day of the week information is acquired either by checking a calendar or querying memory for the current day. When these steps complete, an inference rule is applied to decide the best route. Information acquisition tasks can fail and thus fail to provide a value to a decision-relevant variable— e.g. when no calendar can be found and no information on the current day can be acquired from memory. Similarly, execution may omit an information-acquisition task in response to time-pressure or other factors. In these cases, the inference-rule will rely on a default value — e.g. that today is a weekday.

This framework provides two means for managing resource use in routine decision tasks. First, execution mechanisms can control the timing of resource-demanding tasks by interruption, delaying task initiation, or delaying resumption. For example, if finding a calendar would take perceptual resources (gaze) away from a higher priority task, execution mechanisms can delay this action until the higher priority task completes. Second, since different information-acquisition methods will generally differ in type and amount of required resources, methods can be selected in order to allocate resources them most effectively. For instance, if finding a calendar is prevented by a higher priority use of the gaze resource, execution could try retrieving the information from memory or using the *null-method* (i.e. omitting information acquisition and forcing reliance on a default) which requires minimal resources.

Method-selection in our system is handled by *method-selection-rules* (MSRs) whose syntax mirrors that of the COND macro in LISP. MSRs employ both transient and long-term knowledge. Relevant long-term knowledge can include: the expected interval during which resources must be allocated to a method for it to complete; the usual level of competition for resources from other tasks present during execution of the decision-task (expected workload); the usual amount of time available to complete the decision-task (expected urgency); and the likelihood that the default value associated with an information-acquisition task will prove accurate. Such factors determine the expected utility of alternative information acquisition methods and thus determine a stable preference between alternatives. Taken together, the method preferences for all steps of a decision-method constitute a baseline *decision strategy*.

Transient information can be used to adapt a decision strategy to an agent's current situation. The current model supports adaptation from several kinds of information including especially: *subjective workload* and *default counterevidence*. Subjective workload corresponds to an agent's evaluation of its overall "busyness" compared to expected workload. During periods of unusually high workload for a given decision task, decision strategy is biased in favor of the least resource-demanding methods. In some cases, this will cause decisions to rely on defaults when more reliable information acquisition methods would normally be selected.

Default counterevidence is knowledge that the default value for some decision-relevant factor is likely to prove incorrect for some time into the future. For example, an agent may tend to assume that today is a weekday when deciding a route home from work. Deciding to go into work on a weekend day invalidates this assumption and should (temporarily) reduce the system's tendency to rely on it — i.e. it should bias execution mechanisms to avoid using the default. This function is carried out by task-specific *bias rules*. Since avoiding reliance on valid defaults wastes resources, bias rules must specify a duration, after which their effect expires. The length of this interval depends on the expected duration of the non-default condition and the expected interval between successive observations of the condition if it persists.[1] The system will thus tend to rely on invalid defaults when the default condition lasts for an unusually long time or when an unusually long period has passed since the condition was last observed.

3 Application: user interface evaluation

We have incorporated this approach to routine decision making into action selection mechanisms of our human operator model, APEX.[2] The model consists primarily of two components: action selection mechanisms based on the RAP plan execution system, and a resource architecture which describes limitations on perceptual, cognitive, and motor resources and constrains action selection mechanisms to operate within those limits.

3.1 To err is human, to prevent error is good design

APEX is intended to address a fundamental problem in the design of user interfaces. In particular, newly designed equipment and procedures often inadvertently facilitate human error. Techniques for identifying error facilitations in design tend to be either ineffective or very expensive. For example, one of the most effective ways to test new

[1] See [Freed97] for information on how the bias duration parameter is set.

[2] Architecture for Procedure Execution

designs is to hire human operators to carry out tasks using prototyped equipment, and then observe their performance in a wide range of operating conditions. In our domain, air traffic control, such tests typically require hiring highly paid expert controllers as subjects, often for extended periods. The limited amount of testing that results from high cost can stifle innovation and compromise safety.

One way to get some of the benefits of a "human in the loop" study at much lower cost is to use a computer to simulate all elements of such a study including the equipment, human operators, and experimental observers. Human simulation has been used successfully by others to guide design (e.g. [John90,Corker95]). However, ours appears to be the first system to employ the powerful and versatile action selection mechanisms provided by AI plan execution systems, and thus the first able to function effectively in inherently complex, dynamic, and uncertain domains such as air traffic control. By employing action selection mechanisms designed for robot control, our model overstates human capabilities in some ways, but can operate in domains where predicting human error would be most useful.

Though not specifically designed to make errors of any kind, our approach to managing the resource cost of routine decision-making enables APEX's plan execution component to help predict a type of error sometimes referred to as a "habit capture" [Reason90]. Habit captures are defined by their apparent cognitive cause. In particular, people make such errors when, instead expending resources to acquire reliable information, they act in accordance with a false but usually reliable default assumption. Habit captures are reported quite frequently in naturalistic studies of error [Reason82]. For example:

> "I went to the bedroom to change in to something more comfortable for the evening, and the next thing I knew I was getting into my pajama trousers, as if to go to bed.
>
> "I had decided to cut down my sugar consumption and wanted to have my cornflakes without it. But the next morning, however, I sprinkled sugar on my cereal just as I always do."

In our view, much of people's tendency to rely on default assumptions can be explained as adaptations to regularities in the task environment. For instance, people will be more likely to rely on a default if, apriori, it is especially likely to be true or if the environment reliably provides default counterevidence when it is false. This view provides a basis for predicting when people will rely on false defaults and make habit capture errors as a result.

3.2 An Example

At a TRACON air traffic control facility, one controller will often be assigned to the task of guiding planes through a region of airspace called an arrivals sector. This task involves taking planes from various sector entry points and getting them lined up at a safe distance from one another on landing approach to a particular airport. Some airports have two parallel runways. In such cases, the controller will form planes up into two lines.

Occasionally, a controller will be told that one of the two runways is closed and that all planes on approach to land must be directed to the remaining open runway. A controller's ability to direct planes exclusively to the open runway depends on remembering that the other runway is closed. How does the controller remember this important fact? Normally, the diversion of all inbound planes to the open runway produces an easily perceived reminder. In particular, the controller will detect only a single line of planes on approach to the airport, even though two lines (one to each runway) would normally be expected.

However, problems may arise in conditions of low workload. With few planes around, there is no visually distinct line of planes to either runway. Thus, the usual situation in which both runways are available is perceptually indistinguishable from the case of a single closed runway. The lack of perceptual support would then force the controller to rely on memory alone, thus increasing the chance that the controller will accidentally direct a plane to the closed runway.

3.3 Simulation

When the simulated controller hears that the left runway is closed, interpretation mechanisms cause a propositional representation of this fact to be encoded in memory. The encoding event generates bias (default counterevidence) according to the following rule:

```
(bias-rule-17
 (if (closed ?rwy)
     (create-bias method-27 step5 (10 minutes))))
```

Newly generated bias is represented explicitly in memory along with a notation indicating when the bias will expire if not renewed. In this case, bias lasting 10 minutes causes decision mechanisms to consider the possibility of runway closure (step5 below) in cases where the usual state — all runways open — might otherwise be assumed.

When a plane approaches its airspace, the simulated controller initiates a routine plane-handling method involving accepting responsibility for the plane, determining where the plane is headed, and then guiding it to its destination. If the plane's destination is Los Angeles airport (LAX), guiding it to its destination will involve selecting between the airport's two parallel runways. For highly routine decisions such as runway selection, human controllers can reasonably be expected to know which factors to consider in making the decision and how to appropriately weight each factor. This knowledge is incorporated into the following decision method:

```
(Method-27 (select-runway ?plane)
  (step1 (id-rwy-with-fewer-planes => ?fewer))
  (step2 (id-rwy-fastest-approach ?plane =>
       ?fastest))
  (step3 (id-rwy-easiest-for-me => ?easiest))
  (step4 (id-rwy-better-microclimate =>?climate))
  (step5 (id-available-runways => ?available))
  (step6 (id-safest-rwy ?plane => ?safest))
  (step7 (infer rule-19 ?fewer ?fastest …)
      (wait-for step1 step2 … step6)))
```

In most cases, more than one method will be available for acquiring information about a factor. In this example, the controller could determine runway availability by retrieving information from memory, asking another controller, or by assuming the most likely condition — that the runway is open. Since runways closures are rare and memory retrieval is expensive [Carrier95;Stein93], the decision strategy underlying this method (along with associated MSRs) prescribes reliance on the default assumption unless transient bias (default counterevidence) promotes a more effortful alternative.

In the described scenario, bias produced after learning of the runway closure causes the agent to temporarily avoid reliance on the default. Instead, for some time thereafter, the runway's availability is verified by retrieving information from working memory whenever a runway selection task occurs. Eventually, the initial bias expires. To select a runway for a newly arrived plane, the agent's decisions will once again conform to the default assumption. Other factors will then determine which runway is selected. For example, the controller may choose to direct a heavy plane to the longer left runway which, in normal circumstances, would allow the plane an easier and safer landing. With the left runway closed, actions following from this decision result in error.

Avoiding error requires maintaining appropriate bias. In a variation of the described scenario in which no error occurs, visually perceived reminders of the runway closure cause bias to be periodically renewed. In particular, whenever visual attention mechanisms attend to plane icons on an approach path to the airport, interpretation mechanisms note the absence of a line of planes to the left runway and signal an expectation failure.

```
(expectation-generation-rule-5
  (if (and
        (not (visual-group plane-icons
                                  lft-rwy-path))
        (visual-group plane-icons rt-rwy-path))
     (assert-anomaly (rwy-imbalance right))))
```

In general, whenever an expectation failure occurs, a task to explain the observed anomaly is initiated. The first step in such a task is to try to match the anomaly to a known explanation-pattern (XP) [Schank86]. A match results in a task to verify the XP hypothesis.

```
(explanation-pattern-5
   (:anomaly  (rwy-imbalance ?rwy))
   (:candidate-explanation  (closed ?rwy)))
```

In principle, verifying a hypothesis could involve mental and physical actions of any kind. In the current model however, the only way to verify a hypothesis is to check for a match in working memory. In this case, the contents of working memory are adequate; the anomalous absence of planes on approach to the left runway is explained as a result of the left runway's closure.

Bias renewal occurs whenever the working memory item that originally produced the bias is re-encoded or retrieved. Thus, retrieving the proposition (closed runway left) triggers the bias generation rule just as if the proposition had been encoded for the first time. Thus, the unusual arrangement of planes on the radar scope acts as a constant reminder, preventing the agent from reverting to the use of its default assumption and thereby preventing error.

3.4 Aiding user interface design

Interface designers often overlook aspects of an interface that facilitate user errors, though in many instances, design problems are obvious once pointed out [Norman88]. The problem of noticing these design problems becomes especially difficult in domains such as air traffic control where interfaces must mediate complex tasks carried out in diverse operating conditions. By employing a model of error-prone human behavior, we hope to partially automate the process of predicting design-facilitated errors.

The basis for these predictions arises from an analysis of how agents generally, and humans in particular, can manage limited resources in decision-making. Employing this analysis, it is possible not only to simulate the influence of interface attributes on human tendencies to err, but also to provide causal explanations for predicted errors that indicate ways to repair the design. For example, explaining the described error scenario to a designer as an indirect consequence of low workload indicates a clever fix: runway closures can be visually signaled, but only in low workload conditions when the added screen clutter would cause little distraction.

4 Discussion

The process of deciding action can make substantial demands on limited computational and physical resources. To cope with time-pressure and competing demands, our system decides action on the basis of flexible strategies incorporated into RAP-like reusable plans and other structures. When carried out by the same task execution mechanisms used to carry out non-decision tasks, these decision plans provide two means for managing resources.

First, execution mechanisms can delay or interrupt decision subtasks to give higher priority tasks preferential access to resources. Second, execution can select between alternative methods for subtasks on the basis of duration or compatibility between their different resource requirements and demands from other tasks. Selecting a method for its resource-demand characteristics will sometimes entail trading off against some other desirable attribute such as reliability. In particular, execution may rely on a default assumption in making a decision rather than engage in time- and resource-demanding efforts to acquire more reliable information.

Resource-management strategies that prescribe reliance on defaults make an agent vulnerable to habit-capture errors when assumptions underlying the strategy do not hold in its current environment. In the described example, the strategy of assuming runway availability in the absence of default counterevidence makes the simulated air traffic controller vulnerable in conditions of low workload when counterevidence is unavailable.

The systematicity of such errors makes our approach useful for predicting circumstances in which aspects of an interface design might inadvertently facilitate error. By alerting designers to the potential for such errors early in the design process, we hope to reduce the cost of evaluation and thereby speed the safe introduction of new technology.

Acknowledgements

Thanks to Jim Johnston, Eric Ruthruff, Mark van Selst, and Mike Shafto for many useful discussions.

References

[Carrier95] Carrier, L.M. and Pashler, H. Attentional limitations in memory retrieval. *Journal for experimental psychology: learning, memory, & cognition*, 21, 1339-1348, 1995.

[Corker93] Corker, K.M. and Smith, B.R. An architecture and model for cognitive engineering simulation analysis. *Proceedings of the AIAA Computing in Aerospace 9 Conference*, San Diego, CA, 1993.

[Firby89] R.J. Firby. Adaptive execution in complex dynamic worlds. Ph.D. thesis, Yale University, 1989.

[Freed97] Freed, M. and Shafto, M. Human-system modelling: some principles and a pragmatic approach. *Proceedings of the Fourth International Workshop on the Design, Specification, and Verification of Interactive System*. Granada, Spain.

[Gat96] Gat, Erann. The ESL User's Guide. Unpublished. Available at: www-aig.jpl.nasa.gov/ home.gat/esl.html

[John94] John, B.E. and Kieras, D.E. The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. Carnegie Mellon University. School of Computer Science, TR CMU-CS-94-181, 1994.

[Norman88] Norman, Donald A. *The Psychology of Everyday Things*. Basic Books, New York, N.Y., 1988.

[Pell97] Pell, B., Bernard, D.E., Chien, S.A.., Gat, E., Muscettola, N., Nayak, P.P., Wagner, M., and Williuams, B.C. An autonomous agent spacecraft prototype. *Proceedings of the First International Conference on Autonomous Agents*, ACM Press, 1997.

[Reason90] Reason, J.T. *Human Error*. Cambridge University Press, New York, N.Y., 1990.

[Reason82] Reason, J.T. and Mycielska, K. *Absent-minded? The psychology of mental lapses and everyday errors*. Englewood Cliffs, N.J., Prentice Hall, 1982.

[Schank86] Schank, Roger C. *Explanation Patterns*. Lawrence Earlbaum Associates, Hillsdale, N.J., 1986.

[Simmons94] Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*. 10(1), 1994.

[Stein93] Stein, Earl S. and Garland, Daniel. Air traffic controller working memory: considerations in air traffic control tactical operations. FAA technical report DOT/FAA/CT-TN93/37, 1993.